# A DSP based H.264/SVC decoder for a multimedia terminal

Fernando Pescador, Eduardo Juarez, Mickaël Raulet, César Sanz

# A DSP Based H.264/SVC Decoder for a Multimedia Terminal

F. Pescador, *Member IEEE,* E. Juarez, *Member IEEE,* M. Raulet, *Member IEEE,* and C. Sanz, *Member IEEE*

**Abstract** — *In this paper, the implementation of a DSP-based video decoder compliant with the H.264/SVC standard (14496-10 Annex G) is presented. A PC-based decoder implementation has been ported to a commercial DSP. Performance optimizations have been carried out improving the initial version performance about 40% and reaching real time for CIF sequences. Moreover, the performance has been characterized using H.264/SVC sequences with different kinds of scalabilities and different bitrates. This decoder will be the core of a multimedia terminal that will trade off energy against quality of experience*[1].

**Index Terms — Scalable Video Coding, H.264/SVC, DSP algorithm optimization, performance characterization.**

## I. Introduction

In the last years, a speed-up in the deployment of all kinds of telecommunication networks supporting multimedia services and applications has been produced in many parts of the world. In this context, the consumer multimedia terminals play a central role. In these terminals, video decoding is one of the most demanding tasks in terms of computational load and energy consumption.

The Scalable Video Coding (SVC) techniques [1] can be used in multimedia terminals to achieve a trade-off between quality and energy consumption. Though SVC techniques have been defined in most video coding standards [2][3][4], the SVC capabilities included in H.264 [4] have overcome the ones in former standards.

In PccMuTe[1] project, our research is focused on the energy and power consumption control in multimedia terminals. A multimedia terminal prototype with a DVB-H receiver, an H.264/SVC decoder and an audio decoder is going to be used to validate the experiments. In this context, the H.264/SVC decoder will be used to achieve a trade-off between user Quality of Experience (QoE) and energy consumption [5]. The multimedia terminal architecture is based on a commercial chip [6] having a General Purpose Processor (GPP) and a Digital Signal Processor (DSP). The GPP implements the user interface using a generic operating system and the DSP decodes the video streams.

[1] This work was supported by the Spanish Ministry of Science and Innovation under grant TEC2009-14672-C02-01 (PccMuTe: Power Consumption Control in Multimedia Terminals).

F. Pescador, E. Juarez and C. Sanz, are with the Electronic and Microelectronic Design Group (GDEM) at the *Universidad Politécnica de Madrid*, Spain. (e-mail: {pescador, ejuarez and cesar}@sec.upm.es).

M. Raulet is with *the Institut d'Electronique et de Télécommunications*. De Rennes (IETR)/ INSA Rennes, France (mraulet@insa-rennes.fr).

Up to now, the available SVC decoder implementations are restricted to the PC domain [7][8]. In this work, the Open SVC decoder [8] has been ported to the DSP environment and the methodologies proposed in [9][10][11] to reduce its decoding time have been applied. The real-time performance has been reached for CIF sequences. Up to the best of our knowledge, no other H.264/SVC decoder implementation based on DSP has been reported.

In this paper, a DSP implementation of a real-time H.264/SVC decoder is explained. The H.264/SVC standard and the Open SVC decoder are outlined in Section II. In Section III, the DSP decoder implementation is described. In Section IV the test-bench created to measure the decoder performance is outlined. The results of the profiling tests are discussed in Section V. Finally, Section VI concludes the paper.

## II. The H.264/SVC Standard

In this section, the H.264 standard and the Open SVC decoder are briefly explained for reference.

### A. H.264/SVC Standard

An SVC algorithm was standardized as the annex G of H.264 [4][7] to cover the needs of scalability. In this standard, the video compression is performed by generating a unique hierarchical bit-stream structured in several levels or layers of information, consisting of a base layer and several enhancement layers. The base layer provides basic quality. The enhancement layers provide improved quality at increased computational cost and energy consumption. Because the energy consumption depends on the particular layer to decode, an H.264/SVC decoder is a very well-suited solution for managing the energy consumption by selecting the appropriate layer.

H.264/SVC specifies three types of scalabilities: spatial, temporal and quality. In a temporally scalable video sequence, several frame rates (temporal layers) of a video sequence can be chosen when decoding. Fig. 1 shows an example of a Group of Pictures (GOP) where the user can select three frame rates. If the device decodes the four frames of the GOP (I1, B1, B2, B3), a full-frame-rate sequence will be obtained. If the decoder discards B1 and B3 frames and only decodes I1 and B2, a half-frame-rate sequence will be achieved. The third case is a quarter-frame-rate sequence, which will be obtained when the decoder discards B1, B2 and B3 frames and only decodes I1.

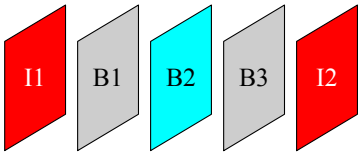


**Fig. 1. Example of a GOP in a temporally scalable bit-stream.**

In a spatially scalable video sequence, several spatial resolutions (spatial layers) of the video frames can be chosen when decoding. Fig. 2 depicts an example of a spatial scalable bit-stream containing three possible resolutions. As can be seen, the information related to the three resolutions of a frame is contained in the field reserved for such frame in the bit-stream.

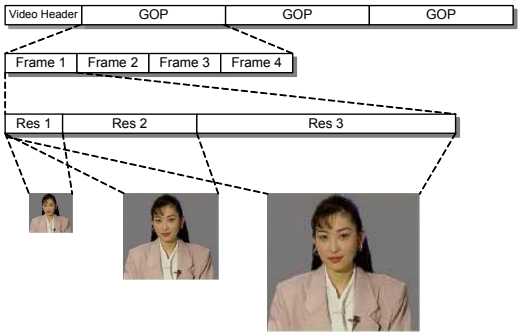


**Fig. 2. Example of a spatially scalable bit-stream.**

In a quality-scalable video sequence (or Signal to Noise Ratio –SNR- sequence), it is possible to select several quality levels (quality layers) when decoding. Fig. 3 shows an example of a quality scalable bit-stream with three qualities. The information related to the three qualities of a frame is contained in the space reserved for this frame in the bit-stream.

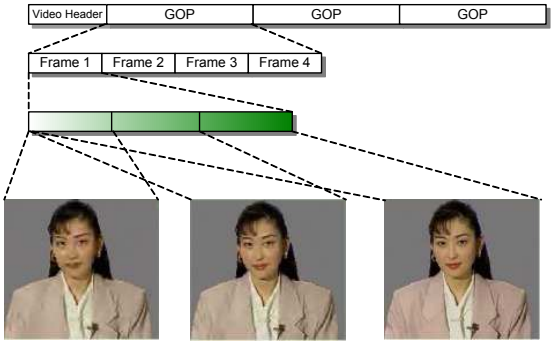


**Fig. 3. Example of a quality-scalable (SNR) bit-stream.**

Finally, the three types of scalability specified in H.264/SVC can be combined into a unique bit-stream. As an example, consider an encoded video sequence that has three temporal layers, three spatial layers and three quality layers. An H.264/SVC decoder that has a medium charged battery may decode, for instance, the third spatial layer to get full spatial resolution, the second temporal layer to get half temporal resolution and the first quality layer to get a low-quality level. A decoder that has a fully charged battery might decode the complete bit-stream to get the full temporal and spatial resolution as well as the higher quality.

### *B. The Open SVC Decoder*

Open SVC Decoder has been developed within [12] from scratch in C language to be easily deployed over embedded systems. It is a flexible library [8] compliant with Scalable Baseline profile, it also encompasses tools allowing flexibility to easily deal with spatial, temporal and fidelity scalability changes. At the beginning it was based on a fully compliant H.264/AVC Baseline library with most of Main profile tools. Only interlaced coding and the weighted prediction are not supported because of their complexity for embedded systems.

Contrary to the JSVM which decodes the upper layer of a given scalable bit-stream, i.e. the enhancement layer with the highest spatial, temporal and quality scalability, the Open SVC Decoder can partially decode the bit-stream until a specific layer is required with a specific temporal scalability. This particularity provides an adaptability of the decoder over different platforms by selecting the right layer in order to have a real-time decoding.

The library contains also several mechanisms to switch of layer during the decoding process which allows the user to select the layer to display by specifying commands. In the case of a partial decoding of a bit-stream, the decoder will dismiss discardable layer. Fig. 4 shows the dataflow graph of the decoding process when the top layer of a 4-layer stream is not decoded. Variable Length Coding and Texture Decoding are processes for the first three layers but not for the fourth.

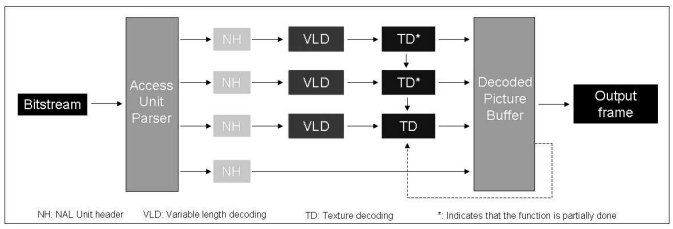


**Fig. 4. Dataflow of decoding process.**

The PC-version of the Open SVC Decoder has been compared to the JSVM 9.19 to benchmark and to test the conformance of the library using conformance test sequences. The benchmarks were executed on a PC with a dual core processor at 2.4GHz and show the speed up between the Open SVC Decoder and the JSVM decoder on several conformance test sequences with different configurations. Indeed, the performance of the library is up to 14 times faster than the JSVM decoder [8].

In Fig. 5, a simplified flow chart diagram of the decoding process for an H.264/SVC compliant bit-stream is shown. The decoder reads the H.264/SVC bit-stream from an input buffer and decodes the NAL units in sequence. After decoding the NAL header, the NAL unit content is identified as a slice header or another syntax element (i. e. a Sequence Parameter Set –SPS– or a Picture Parameter Set –PPS–). When the NAL unit contains a slice of interest for the selected layer, the decoder extracts all the syntactical elements from the bit-stream and stores them in intermediate buffers. If the

processed NAL must be displayed, each macroblock (MB) is completely decoded, however, if the NAL must not be displayed the MB is partially decoded.

In the next step, if a frame has been completely decoded, the deblocking filter is applied. Finally, the decoded pictures are stored in images buffers and presented in the right order using the PC Simple Direct Media Layer (SDL) library.

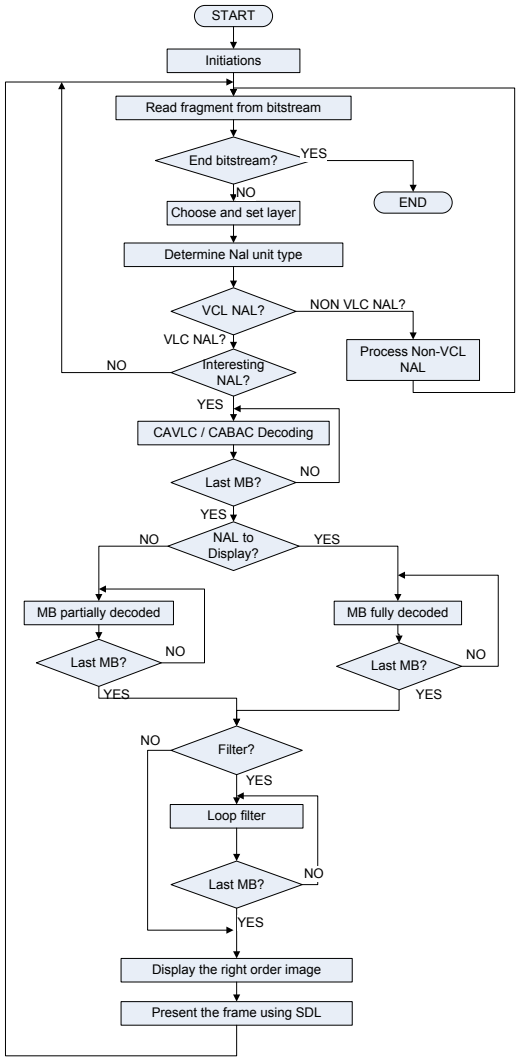


**Fig. 5. Simplified Open SVC decoder flow chart.**

## III. DSP IMPLEMENTATION

### *A. Processor Architecture*

The processor [5] basically consists of two processing cores, a GPP and a DSP. The former processor [13] is aimed to run a generic Operating System (OS) while the latter [14] has an architecture optimized for video processing.

The GPP processor has two levels of cache memories (L1 and L2). The program (L1P) and data (L1D) caches, within the Microprocessor Unit (MPU) Subsystem, consist of a 16 KB memory space. The L2 cache consists of a 256 KB memory space, shared between program and data. In addition, the MPU integrates a coprocessor, optimized for multimedia applications, with its own multiplication-accumulation unit (MAC) and support for floating-point operations.

The fixed-point DSP core has two levels of internal memory (L1 and L2). The L1P memory/cache consists of a 32 KB memory space and the L1D memory consists of an 80 KB memory space. Both memories can be configured as cache memories, general-purpose memories or a combination of both. Finally, the L2 memory/cache consists of a 64 KB memory space, shared between the program and data. L2 memory can be configured as a general-purpose mapped memory, a cache memory, or a combination of both.

A commercial prototyping board [15] (Fig. 6) based on this processor has been used to test the Open SVC decoder and its performance has been measured. The board has 256 MB of SDRAM external memory, 256 MB of Flash external memory and several interfaces. Note that the clock frequency of the GPP and DSP cores is 600 MHz and 500 MHz, respectively.

**Fig. 6. Prototyping system based on the commercial processor.**

### *B. Open SVC Decoder Porting Process*

It is worth noting that the Open SVC decoder has been developed for a PC-based platform. The decoder has been ported to the DSP as follows:

- The decoder has been encapsulated into a Real Time Operating System (RTOS) [16] task executed by the DSP. The size of the stack associated to this task has been adjusted to 1 MB and has been allocated in external memory. Code and data have been allocated in external memory.

- To limit the amount of memory of the DSP implementation, the decoder code has been modified and the maximum size of the decoded pictures has been reduced from HD (1920×1080) to SD (720×576).

- Internal memory has been configured as follows: L1D is divided in 32 KB for cache memory and 48 KB for general purpose data; L1P is configured as a 32 KB cache program memory and L2 is splitted between level-2 cache memory and general purpose memory. Currently, neither the code

nor the data are allocated in internal memory but these memories have available space for future optimizations.

- The decoder output interface has been modified. In the original code, the decoded pictures are displayed on screen using the SDL library. In the DSP code, the decoded pictures are written in a YUV file.
- Functions used to access the bit-stream files have been adapted to the functions available in the DSP real-time support libraries.
- The way to select the layer to be decoded has been modified. In the original code, the layer was selected using the command line arguments while in the DSP version these parameters are introduced through a configuration file that is parsed at the beginning of the decoding process.

### C. *Optimization Process*

The performance of the DSP-based decoder has been measured using several standard sequences and the manufacturer profiling tools. The modules having the highest computational load have been identified. The methodologies presented in [9][10][11] have been applied to reduce the number of CPU cycles needed to decode an H.264/SVC bit-stream. These methodologies improve the decoder performance taking advantage of the SIMD (Simple Instruction Multiple Data) architecture and using explicit DMA transfers to move data between internal and external memory.

The SIMD architecture allows operating with several pels/coefficients at the same time using assembly instructions. As an example of the use of these instructions, the algorithm implemented to calculate the interpolated pels when the motion vectors have a ¼-pel resolution is summarized.

To calculate the interpolated pels a 6-tap filter must be applied in horizontal and vertical directions. The Fig. 7 shows the pels used to calculate the intermediate values for an 8x8 pel block (shadowed pels) with a fractional motion vector only in the horizontal direction. Thirteen pels (from $p_{-2}$ to $p_{10}$) must be read for each row. This pels are stored in four 32-bit variables (Data1 to Data4) using 4 double-word load instructions.

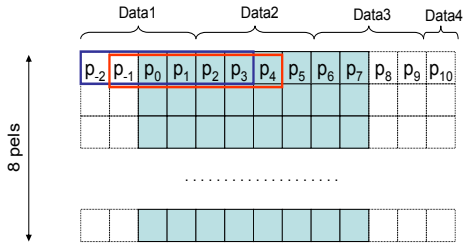


**Fig. 7. Pels used to calculate the interpolated pels of a row.**

Fig. 8 summarizes the optimized algorithm used to calculate the interpolated pels for the first row. The first interpolated pel ($I_0$) uses the pels stored in Data1 and Data2 (from $p_{-2}$ to $p_3$). Each pel is multiplied by a constant coefficient, all the results are accumulated and finally the average is calculated. All these operations can be optimized using a specific SIMD instruction available in the DSP (_dotpsu4). Two of these instructions are needed and the final results are stored in two 16 bits variables ($M_1$ and $M_3$). The average between both data must be calculated as will be show later.

The second interpolated pel ($I_1$) is calculated using the same algorithm but in this case employ the pels from $p_{-1}$ to $p_4$. All these pels are stored in Data1 and Data2 so no additional loads are necessary. The results for this pel are stored in $M_2$ and $M_4$ variables.

The $M_i$ results are stored in 16 bits variables. Package instructions can be used to store two of them in a 32-bits variable. $M_{12}$ and $M_{34}$ variables store the packed data. Using the addition (_add2) and right shift (_shr2) instructions is possible to calculate the average between $M_1$ and $M_3$ and, $M_2$ and $M_4$, at the same time. Interpolated pels $I_0$ and $I_1$ are obtained after these operations with 16-bit resolution and packed in a 32-bit variable. These interpolated pels must have a resolution of 8 bits, so four pels (from $I_0$ to $I_3$) can be packed into a 32-bit variable. After the optimization process, the algorithm needs only 12 instructions (2 loads, 4 products, 3 packages, 1 addition, 1 shift and 1 store).

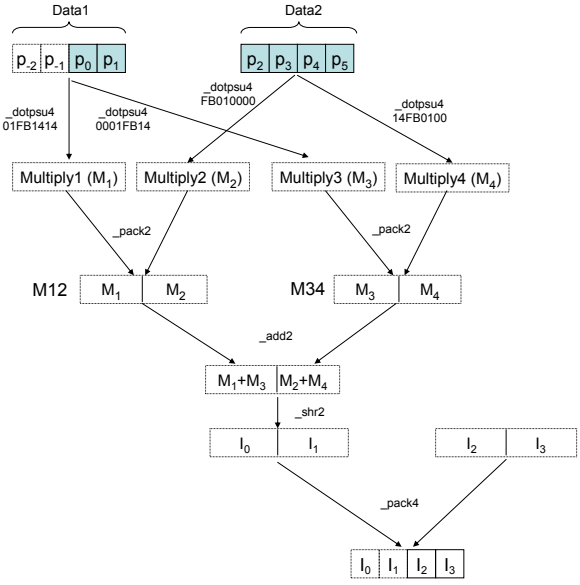


**Fig. 8. Algorithm implemented to calculate the interpolated pels with ¼ pel resolution in horizontal direction.**

A similar methodology have been applied to CABAC entropy decoding, frames upsampling, motion compensation, IICT, coefficients interpolation and deblocking filter. In Subsection V.A, Table I presents the average improvement achieved in all the optimized modules.

Moreover, the DMA controller has been used to improve the data transfers between internal and external memory during the motion compensation process. The data used in the MB loop ("MB fully decoded" block in Fig. 5) are allocated in internal memory to increase the execution speed. The reference data pointed by the motion vectors are moved from the reference picture buffers to a buffer in internal memory (REF_Y). The prediction is added with the residual MB and

stored in a ping-pong buffer (REC). To move the reference and reconstructed data from/to external memory to/from internal memory, explicit DMA transfers are used. The Fig. 9 shows the buffers allocated in internal memory for the motion compensation process.

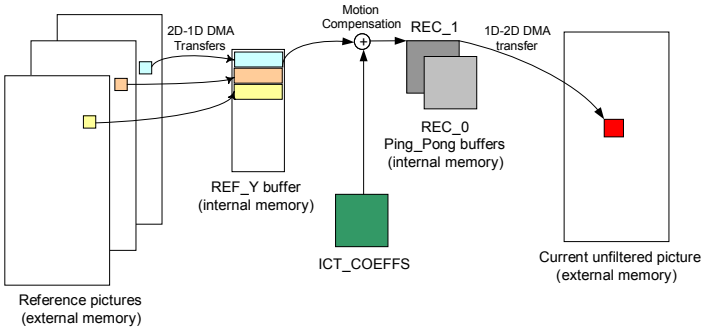


**Fig. 9. Transfers between internal and external memory and use of internal buffers to decode one MB.**

## IV. TEST-BENCH

A set of tests has been carried out to verify the decoder conformance and to characterize its performance using different combinations of scalability values and bitrates. A block diagram of the test-bench is shown in Fig. 10. As can be seen, first, a test stream is read from a file and written into a stream buffer allocated in external memory. Then, the decoder reads the stream from the memory and decodes it on a picture basis. At last, the decoded picture is written into a buffer and also into a component YUV video file. The test-bench has been executed in the prototype board used in PccMuTe project (see Fig. 6).

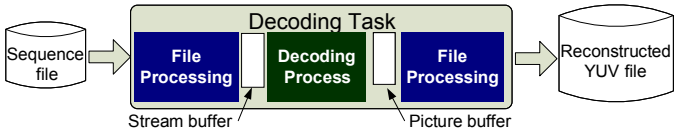


**Fig. 10. Test-bench block diagram to profile the Open SVC decoder in real time.**

In order to assess the decoder performance with the test-bench depicted in Fig. 10, six well-known video sequences (Akiyo, Coastguard, Flower, Foreman, Mobile and News) have been encoded using a commercial H.264/SVC encoder [17]. The following subsections summarize the generated sequences.

### A. Performance Dependence on Scalability

Two different types of test sequences have been generated to evaluate the influence of the specific layers embedded on the stream in the decoder performance. For each type of set, sequences that consist of six layers extracted out from the eight possible combinations among two spatial resolutions (QCIF and CIF), two frame-rates (12.5 and 25 frames per second) and two qualities (low and high) have been generated. Furthermore, the bitrate of these sequences is 512 Kbps and the base layer of each sequence has been encoded with 102 Kbps (20% of a total bitrate of 512 Kbps).

The stream structure of the first set of test sequences, exemplified with the *Akiyo* sequence, can be seen in Fig. 11. Note that the two possible temporal scalability values are omitted. In this type of test sequence, the first enhancement layers are derived from the corresponding base layers with only an increase in quality while the second enhancement layers are derived from the previous ones with only an increase in spatial resolution. In this paper, they are designated as *quality-spatial* sequences to stress the fact that the greatest quality layer is obtained from the base layer with, first, a quality improvement and, then, with a spatial resolution improvement.


**Fig. 11. Quality-Spatial six-layered test sequence structure. – temporal resolution omitted.**

Fig. 12 shows the stream structure of the second set of test sequences. The first enhancement layers are derived from the base layers with only an increase in spatial resolution although the second enhancement layers are generated from the first enhancement ones with an increase in quality. In the rest of the paper the sequences belonging to this set are designated as *spatial-quality* sequence.


**Fig. 12. Spatial-Quality six-layered test sequence structure.- temporal resolution omitted.**

As far as the codec parameters to generate the test sequences concern, the GOP size equals 8 progressive frames, the CABAC is used for entropy coding, the deblocking filter is active, all possible macroblock partitions are enabled for inter-prediction, three reference frames are allowed, and one B-frame is coded for each I-frame.

The decoder performance results using the previous sequences are presented in Table III and Table IV and they are discussed in Subsection V.B.

### B. Performance Dependence on Bitrate

In addition, a set of sequences has been generated in order to evaluate the influence of the bitrate in the decoder

performance. The *foreman* sequence has been selected to analyze the dependency between the bitrate and the decoder performance. The sequence has been encoded as a quality-spatial stream using the codec parameters described in Subsection IV.A. Table V provides the performance results for three different bitrates (0.5 Mbps, 1 Mbps and 2 Mbps) and in Subsection V.C some conclusions are derived.

## V. RESULTS

This section describes the decoder performance, measured as the number of CPU cycles employed to decode a frame of a sequence layer, after the optimization process. While in Subsection V.A, the performance improvement of each optimized module is summarized, in Subsection V.B the decoder performance is analyzed using quality-spatial and spatial-quality sequences. Finally, Subsection V.C presents the decoder performance using sequences with different bitrates.

### *A. Decoder Modules Improvement after optimization*

The optimization techniques presented in Subsection III.C have been applied to the DSP-based decoder implementation. Each layer of the sequences described in Subsection IV.A has been decoded and the profile data of each module has been analyzed.

Table I presents the average performance improvement achieved in each of the optimized modules for the *foreman* quality-spatial sequence. The entry "others" in Table I includes functions optimized for bit-stream parsing, intra-prediction and motion vectors storage. To obtain these measurements, each layer of the sequence *foreman*, encoded with the parameters presented in Subsection IV.A, has been decoded with a decoder that encompasses all optimized versions of the modules shown in Table I.

**TABLE I**
**OPTIMIZED MODULES AND AVERAGE PERFORMANCE IMPROVEMENT FOR THE FOREMAN SEQUENCE.**

| Module | Performance improvement |
|---|---|
| CABAC entropy decoding | 59.0% |
| Deblocking filter | 28.3% |
| Motion compensation & interpolation | 72.3% |
| Inverse ICT | 80.1% |
| Coefficients Scalability & SNR | 58.7% |
| Others | 58.6% |

In Table II, the average performance improvement percentage per module and sequence layer is shown. These values have been obtained as follows. First, different optimized decoder versions that comprehend optimizations for only one module have been generated. Afterwards, each layer of the quality-spatial *foreman* sequence, similarly encoded with the parameters presented in Subsection IV.A, has been decoded with each optimized decoder. At last, the average number of CPU cycles per layer frame is compared to that of the non-optimized decoder.

The columns of Table II present the layers included in the sequence where S indicates the picture size of the frames, T the temporal resolution in frames per second and Q the level of quality (high or low). The rows present the percentage of improvement achieved when an optimized module is integrated. Finally, the last row shows the global improvement when all the optimized modules are integrated.

**TABLE II**
**GLOBAL AND MODULE AVERAGE PERFORMANCE IMPROVEMENT FOR THE QUALITY-SPATIAL FOREMAN SEQUENCE.**

| | Layer 0 S=QCIF T=12.5 Q=Low | Layer 1 S=QCIF T=25 Q=Low | Layer 2 S=QCIF T=12.5 Q=High | Layer 3 S=QCIF T=25 Q=High | Layer 4 S=CIF T=12.5 Q=High | Layer 5 S=CIF T=25 Q=High |
|---|---|---|---|---|---|---|
| CABAC | 11.0 | 7.4 | 17.6 | 12.1 | 16.1 | 10.7 |
| Deblocking Filter | 7.2 | 5.6 | 5.5 | 4.1 | 6.2 | 7.1 |
| MC & Interpol. | 2.8 | 5.2 | 6.3 | 3.2 | 7.0 | 6.4 |
| IICT | 7.0 | 3.6 | 4.9 | 3.5 | 3.4 | 4.8 |
| SNR | 0.8 | -0.3 | 12.4 | 13.8 | 0.4 | -0.1 |
| Others | 1.0 | 1.6 | 3.9 | 5.1 | 6.4 | 6.1 |
| Optimized Version | 35% | 34% | 39% | 38% | 40% | 35% |

The two following conclusions can be drawn from the analysis of Table I and Table II:

First, Table II shows that the global improvement achieved is not the addition of the improvement of each module. The reason of this loss is that the allocation of the modules in memory changes after each optimization and the number of data-cache misses increases when the code is optimized.

Secondly, Table I indicates that some modules have been optimized achieving an improvement greater than 70%. But the global improvement shown in Table II is around 40%. This difference is justified by the flow chart of the decoder presented in Fig. 5. The decoder executes the decoding phases (entropy decoding, MC and deblocking filter) frame by frame generating data cache misses and increasing the number of cycles used to decode each picture. Currently the flow chart is being modified to reduce the cache misses.

Finally, the performance improvement achieved with the "SNR" module for the layers 1 and 5 is negative. This module is only used if a quality enhancement layer is decoded (layers 2 or 3). The rest of the enhancement layers do not use this module so the global improvement should be zero. However, the integration of this module modifies the allocation of the code and the data in memory and therefore the number of cache misses varies. This situation generates a negative improvement in the decoder performance for layers 1 and 5.

The use of the DMA in the motion compensation process achieves an improvement lower than 4% in the global performance. This improvement is smaller than expected because the CPU must wait for the end of transfers before processing the transferred data.

The modifications in the decoder flow chart proposed above to reduce the data cache misses will allow to reduce the CPU waits during the DMA transfers. The CPU will be able to execute some phases of the algorithm while the DMA is transferring data (further details in [10]).

### *B. Decoder Performance with different kinds of scalabilities*

This subsection presents the decoder performance results using the sequences described in Subsection IV.A. The performance is calculated after decoding 100 frames.

First, the decoder performance is measured using the quality-spatial sequences. Table III contains the percentage of CPU cycles needed to achieve real-time processing out of those available, using the un-optimized and optimized decoder versions and for all layers. Moreover, the percentage of improvement achieved for each layer is presented. These results have been obtained using a DSP running at 500 MHz.

The last row presents only for reference the average improvement achieved. These results demonstrate that real-time performance has been achieved for all the layers of the generated streams.

**TABLE III**
**OPEN SVC DECODER PERFORMANCE BEFORE AND AFTER THE OPTIMIZATION PROCESS FOR QUALITY-SPATIAL SEQUENCES.**

| | | Layer 0 S=QCIF T=12.5 Q=Low | Layer 1 S=QCIF T=25 Q=Low | Layer 2 S=QCIF T=12.5 Q=High | Layer 3 S=QCIF T=25 Q=High | Layer 4 S=CIF T=12.5 Q=High | Layer 5 S=CIF T=25 Q=High |
|---|---|---|---|---|---|---|---|
| Akiyo | Unoptim | 10.1 | 20.5 | 22.7 | 50.5 | 60.7 | 127.7 |
| | Optim | 6.5 | 13.5 | 13.7 | 30.9 | 36.1 | 84.1 |
| | Improve | 36% | 34% | 40% | 39% | 41% | 34% |
| Coast Guard | Unoptim | 10.2 | 21.3 | 23.2 | 52.1 | 62.7 | 135.7 |
| | Optim | 6.6 | 14.0 | 13.8 | 31.7 | 37.2 | 86.5 |
| | Improve | 35% | 34% | 40% | 39% | 41% | 36% |
| Flower | Unoptim | 9.7 | 20.6 | 22.5 | 51.2 | 60.6 | 130.7 |
| | Optim | 6.3 | 13.5 | 13.3 | 30.9 | 36.0 | 84.2 |
| | Improve | 35% | 35% | 41% | 40% | 40% | 41% |
| Foreman | Unoptim | 10.3 | 21.5 | 22.8 | 51.7 | 61.3 | 133.1 |
| | Optim | 6.7 | 14.2 | 13.8 | 31.8 | 36.7 | 86.1 |
| | Improve | 35% | 34% | 39% | 38% | 40 | 35% |
| Mobile | Unoptim | 10.7 | 21.4 | 23.1 | 52.2 | 63.8 | 132.6 |
| | Optim | 6.9 | 14.0 | 14.1 | 31.9 | 37.6 | 86.0 |
| | Improve | 36% | 35% | 39% | 39% | 41% | 35% |
| News | Unoptim | 9.9 | 20.5 | 22.6 | 49.6 | 59.8 | 128.0 |
| | Optim | 6.4 | 13.6 | 13.5 | 30.4 | 36.2 | 82.5 |
| | Improve | 35% | 34% | 40% | 39% | 39% | 36% |
| Average Improvement | | 35% | 34% | 40% | 39% | 40% | 37% |

Later, the decoder performance is measured using the spatial-quality sequences. Table IV contains the percentage of CPU cycles needed to achieve real-time processing using the un-optimized and the optimized versions and the percentage of improvement achieved for each layer. The average improvement achieved for each layer is showed in the last row. In this case, the real time performance is not achieved for layer 5.

**TABLE IV**
**OPEN SVC DECODER PERFORMANCE BEFORE AND AFTER THE OPTIMIZATION PROCESS FOR SPATIAL-QUALITY SEQUENCES.**

| | | Layer 0 S=QCIF T=12.5 Q=Low | Layer 1 S=QCIF T=25 Q=Low | Layer 2 S=QCIF T=12.5 Q=High | Layer 3 S=QCIF T=25 Q=High | Layer 4 S=CIF T=12.5 Q=High | Layer 5 S=CIF T=25 Q=High |
|---|---|---|---|---|---|---|---|
| Akiyo | Unoptim | 9.4 | 19.9 | 47.7 | 106.0 | 89.8 | 208.6 |
| | Optim | 6.1 | 13.2 | 28.4 | 68.3 | 51.5 | 122.6 |
| | Improve | 35% | 34% | 40% | 36% | 43% | 41% |
| Coast Guard | Unoptim | 10.1 | 21.2 | 51.4 | 110.5 | 92.4 | 208.7 |
| | Optim | 6.6 | 13.9 | 30.9 | 70.8 | 52.9 | 126.0 |
| | Improve | 35% | 34% | 40% | 36% | 43% | 40% |
| Flower | Unoptim | 10.0 | 20.3 | 51.1 | 109.9 | 91.8 | 206.7 |
| | Optim | 6.5 | 14.1 | 30.0 | 69.9 | 51.1 | 126.0 |
| | Improve | 35% | 31% | 41% | 36% | 44% | 39% |
| Foreman | Unoptim | 10.1 | 21.6 | 50.3 | 110.7 | 91.0 | 109.2 |
| | Optim | 6.6 | 14.2 | 30.5 | 70.6 | 52.7 | 126.4 |
| | Improve | 35% | 34% | 39% | 36% | 42% | 40% |
| Mobile | Unoptim | 10.3 | 21.3 | 51.6 | 109.2 | 93.4 | 211.5 |
| | Optim | 6.6 | 13.9 | 31.0 | 71.0 | 54.0 | 125.5 |
| | Improve | 36% | 35% | 40% | 35% | 42% | 41% |
| News | Unoptim | 9.5 | 20.0 | 48.1 | 106.7 | 90.1 | 206.4 |
| | Optim | 6.2 | 13.3 | 29.1 | 68.7 | 50.9 | 124.1 |
| | Improve | 35% | 34% | 39% | 36% | 43% | 40% |
| Average Improvement | | 35% | 34% | 40% | 36% | 43% | 40% |

The results presented in Table III and Table IV demonstrate that the performance is higher if the first enhancement layer is a SNR layer instead of a spatial enhancement layer. Real-time performance is achieved for the first subset of sequences presented in Subsection IV.A but not for the second subset.

### *C. Decoder Performance with different bitrates*

Finally, Table V shows the influence of the bitrate in the decoder performance. As described in Subsection IV.B three quality-spatial streams has been generated with different bitrates (0.5, 1 and 2 Mbps) using the same parameters for the encoder configuration. The sequence *foreman* has been used to analyze the decoder performance.

The CPU percentage needed to achieve real time performance is presented for all the layers included in the bit-streams. Moreover, the percentage of increase in the number of CPU cycles needed to decode the 1 Mbps and 2 Mbps streams respect to 0.5 Mbps stream is shown. All the results have been obtained after decoding 100 frames.

**TABLE V**
**RELATIONSHIP BETWEEN BITRATE AND DECODER PERFORMANCE.**

| | | S=QCIF T=12.5 Q=Low | S=QCIF T=25 Q=Low | S=QCIF T=12.5 Q=High | S=QCIF T=25 Q=High | S=CIF T=12.5 Q=High | S=CIF T=25 Q=High |
|---|---|---|---|---|---|---|---|
| 0.5 Mbps | %CPU | 6.7 | 14.2 | 13.8 | 31.8 | 36.7 | 86.1 |
| 1 Mbps | %CPU | 7.7 | 16 | 15.6 | 35.3 | 40.8 | 91.9 |
| | Increase | 14.9% | 12.7% | 13.0% | 11.0% | 11.2% | 6.7% |
| 2 Mbps | %CPU | 9.2 | 18.8 | 18.6 | 40.5 | 46.8 | 104.7 |
| | Increase | 37.3% | 32.4% | 34.8% | 27.4% | 27.5% | 21.6% |

The results presented in Table V show that the bitrate has a higher influence in the decoder performance for the base layer than for rest of the layers. Moreover, the increase in the number of CPU cycles needed to achieve real time

performance is not linear with the bitrate, if the bitrate is doubled; the number of the CPU cycles increase in about 15%.

## VI. Conclusion & Future Work

An H.264/SVC decoder based on a commercial DSP has been implemented by porting the Open SVC decoder from the PC to the DSP environment. Several optimizations techniques have been applied to reach real-time performance for CIF sequences. Up to the best of our knowledge, no other H.264/SVC decoder based on DSP has been reported. This optimized decoder will be used in a multimedia terminal to trade-off between quality and energy consumption.

It is worth noting that the gap between the module and global improvements of the ported decoder is mainly due to data-cache misses and the increasing number of CPU cycles employed at the frame-by-frame decoding phase. Furthermore, the motion compensation process achieves smaller ameliorations than could be expected when the DMA is in used. In addition, the optimized Open SVC decoder accomplishes higher enhancements for *quality-spatial* test sequences than for *spatial-quality* ones. Finally, the main performance decrease at increasing bit rates is observed when decoding base layers.

In near future the work will be focused on two lines. The former consists in the distribution of data and code in the different levels of memory and the flow chart reorganization to reduce the number of cache misses, while the latter will concentrate in evaluating the correlation between the decoded layer and the DSP energy consumption.

## Acknowledgment

The authors would like to thank D. Samper, E. Seisdedos and J. J. Soriano from GDEM-UPM and M. Blestel from INSA for their contributions to this work.

## References

[1] J-R Ohm, “Advances in Scalable Video Coding”. Proceedings of the IEEE, vol. 93, nº 1 pp. 42-56, Jan. 2005.

[2] ISO/IEC 13818-2 (ITU-T Rec. H.262). Generic coding of moving pictures and associated audio information: Video. 1995.

[3] ISO/IEC 14496-2. Information technology. Coding of audio visual objects. Part 2: Video. 1998.

[4] ISO14496-10. Information technology. Coding of audio-visual objects. Part 10: Advanced Video Coding. December 2005.

[5] E. Juárez, F. Pescador, P.J. Lobo, A. Groba, and C. Sanz. “Distortion-Energy Analysis of an OMAP-Based H.264/SVC Decoder” 6th Int. ICST Conference on Mobile Multimedia Communications Sept 2010. Lisbon, Portugal. ISBN: 978-963-9799-98-1.

[6] Texas Instruments. OMAP 3530 Technical Reference Manual. Literature Number: SPRUF98O. April 2010– Revised February 2011

[7] Joint Scalable Video Model JSVM-19, ISO/IEC JTC1/SC29/WG11 ITU-T SG16 Q.6, N9212, 2010.

[8] M. Blestel and M. Raulet. “Open SVC Decoder: a flexible SVC library” International conference on Multimedia 2010, Open Source Software Competition Program. October 2010. Firenze, Italy. Pp 1463-1466.

[9] F. Pescador, C. Sanz, M.J. Garrido, E. Juárez and D. Samper. “A DSP Based H.264 Decoder for a Multi-Format IP Set-Top Box”. IEEE Trans. on Consumer Electronics Vol. 54, Issue 1, February 2008 pp. 145-153.

[10] F. Pescador, G. Maturana, M.J. Garrido, E. Juárez and C. Sanz “An H.264 video decoder based on a DM6437 DSP”. IEEE Trans. on Consumer Electronics. Vol. 55, Nº 1. Pp. 205-212. February 2009.

[11] F. Pescador, D. Samper, M.J. Garrido, E. Juárez and M. Blestel. "A DSP based SVC IP STB using Open SVC Decoder". Int. Symposium on Consumer Electronics. Braunschweig Germany, 7-10. June 2010.

[12] M. Barkowsy, M. Blestel, M. Carnec, A. Ksentini, P. Le Callet, G. Madec, R. Monnier, JF. Nezan, R. Pepion, Y. JF. Travers, M. Raulet, and A. Untersee, “Overview of the svc4qoe project”, 6th Int. ICST Conference on Mobile Multimedia Communications September 2010, Lisbon, Portugal. ISBN: 978-963-9799-98-1.

[13] ARM Limited. Cortex-A8 Technical Reference Manual. Revision r2p1. November 2007.

[14] Texas Instruments, TMS320C64x/C64x+ DSP CPU and Instruction Set, SPRU732H, October 2008.

[15] BeagleBoard System Reference Manual Rev. C4, December 2009.

[16] Texas Instruments. TMS320 DSP-BIOS User's guide SPRU423H. August 2009.

[17] MainConcept. SVC Baseline SDK DirectShow Documentation. Version 1.0.0. Aachen, Septembre 22, 2009.

**Fernando Pescador** (M’07) received the Ingeniero Técnico de Telecomunicación degree in 1992 and the Ingeniero de Telecomunicación degree in 2001, both from the Universidad Politécnica de Madrid (UPM), Spain. He is Associate Lecturer at the Department of Electronic and Control Systems at E.U.I.T. de Telecomunicación of the UPM since 1995 and researcher of the Electronic and Microelectronic Design Group (GDEM) since 1999. His research interests are real time video coding and digital video broadcasting.

**Eduardo Juárez** (M’96) received the Ingeniero de Telecomunicación degree from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 1993 and the Docteur ès Sciences Techniques degree from the École Polytechnique Fédéral de Lausanne (EPFL), Lausanne, Switzerland, in 2003. In 1994, he joined the Digital Architecture Group (GAD) of the UPM as a researcher. In 1998, he joined the Integrated Systems Laboratory (LSI) of the EPFL as an Assistant. In 2000, he joined Transwitch Corp., Switzerland, as Senior System Engineer. In 2004, he joined the Electronic and Microelectronic Design Group (GDEM) as a post-doctoral researcher. In 2007, he joined the faculty of the E.U.I.T. de Telecomunicación of the UPM. His current interests are in the design of low-power video and audio decoders for mobile applications.

**Mickaël Raulet** received the Engineering degree in electronic and computer engineering from National Institute of Applied Sciences (INSA), Rennes Scientific and Technical University. In 2006, he received the Ph.D. degree from INSA in electronic and signal processing in collaboration with the software radio team of Mitsubishi Electric ITE (Rennes–France). He is currently a researcher at the Institute of Electronics and Telecommunications of Rennes (IETR). Since 2007, he has been contributing to the ISO/IEC JTC1/SC29/WG11 (MPEG) standardization activities for the development of the RVC standard.

**César Sanz** (S’87. M’88) received the Ingeniero de Telecomunicación degree with honours in 1989 and the Doctor Ingeniero de Telecomunicación degree with summa cum laude in 1998 both from the Universidad Politécnica de Madrid (UPM). Since 1984 he has been a member of the faculty of the E.U.I.T. de Telecomunicación of the UPM, since 1999 has been Associate Professor at the Department of Electronic and Control Systems and since 2008 he is the director of the E.U.I.T. de Telecomunicación. In addition, he leads the Electronic and Microelectronic Design Group (GDEM) involved in R&D projects with Spanish and European companies and public institutions. His areas of interest are microelectronic design applied to image coding, digital TV and digital video broadcasting.